Machine Learning aided Record Linkage

Group Components

- **Francesco Porto** f.porto2@campus.unimib.it (816042)
- **Francesco Stranieri** f.stranieri1@campus.unimib.it (816551)
- **Mattia Vincenzi** m.vincenzi14@campus.unimib.it (860579)

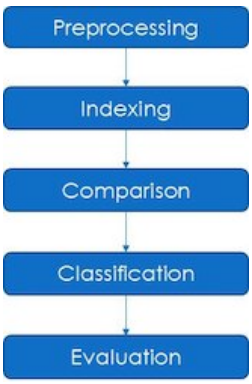
Abstract

Record Linkage is the process of finding records in one or more datasets that refer to the same entity across different data sources. Traditionally, it is done by applying comparison rules between pairs of attributes from each dataset. In this project we investigate some possible Machine Learning applications to Record Linkage (and Data deduplication) by doing **two different experiments**. In the first experiment, we compare two different Indexing methods (Full vs Block) to figure out which one works better for our models in terms of time and quality of results. In the second experiment, we try to investigate the re-usability of the models we create, training on a dataset and testing on the other, and their viability in real-case scenarios.

Python Record Linkage Toolkit

Throughout the project, we make use of a Python library called ["Python Record Linkage Toolkit" (https://github.com/J535D165/recordlinkage/blob/master/docs/index.rst)](https://github.com/J535D165/recordlinkage/blob/master/docs/index.rst), which provides a simple framework to facilitate the process of Record Linkage. In the context of this library, the Record Linkage process is divided into 5 steps:

- **Preprocessing (optional)**: clean the datasets in order to improve the odds of finding matches.
- **Indexing**: create record pairs (R1, R2) such that R1 belongs to the first dataset and R2 belongs to the second one. Such pairs are called "*candidate links*", and can be obtained via different methods (e.g. Full Index or Block Index)
- **Comparison**: compare record pairs via different metrics and thresholds, in order to obtain measures of similarity between each pair. Note that no cardinality reduction is performed in this phase.
- **Classification**: classify record pairs, according to their measure of similarity, into matches or non-matches. It can be done either by using a deterministic approach or different Machine Learning algorithms.
- **Evaluation**: evaluate the results of the Classification phase via Confusion Matrices.



Helper Functions

We have defined a few helper functions that allow us to streamline the Record Linkage process:

- **Indexing_dataset** : given two datasets, returns the result of the indexing process (either *full* or *block*) on the given attributes

- **Compare_records** : given two datasets, a set of rules specifying how to compare some attributes, and a list of attributes that must match exactly, returns a multi index containing the comparisons' results.

- **Evaluate_results** : returns a confusion matrix of a given classification method; also returns its precision, f-score and recall

Experiment 1: Comparison between Full Index and Block Index

In this experiment we present two examples of Record Linkage, the first one using a **Full Index**, that is we make the cartesian product of the records in the first dataset and the records in the second one; the second one uses **Blocking**, that only finds a subset of the cartesian product.

Part 1: Using a Full Index

Dataset description

We use the FEBRL (Freely Extensible Biomedical Record Linkage) dataset since it provides the *true links* for optimal Record Linkage. The paper used as reference is available here (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.481.8577&rep=rep1&type=pdf).

```
In [51]: from recordlinkage.datasets import load_febrl4
```

This dataset contains 10000 records (5000 originals and 5000 duplicates, with one duplicate per original); the originals have been split from the duplicates into dataset4a.csv (containing the 5000 original records) and dataset4b.csv (containing the 5000 duplicate records).

```
Loading data...
5000 records in dataset A
5000 records in dataset B
5000 links between dataset A and B
```

We take a first look at the dataset. It contains auto-generated records about patients. There are 11 fields:

- **rec_id**: the id of the record
- **given_name**: the name of the patient
- **street_number**: the street number of the patient's house
- **address_1**: the road where the patient lives
- **address_2**: the city where the patient lives
- **suburb**: the suburb in the city where the patient lives
- **postcode**: the postcode of the city where the patient lives
- **state**: the state where the patient lives
- **date_of_birth**: the date of birth of the patient
- **soc_sec_id**: the social security number of the patient

rec_id	given_name	surname	street_number	address_1	address_2	suburb	postcode	state	date_of_birth	soc_sec_id
rec-1070-org	michaela	neumann	8	stanley street	miami	winston hills	4223	nsw	19151111	5304218
rec-1016-org	courtney	painter	12	pinkerton circuit	bega flats	richlands	4560	vic	19161214	4066625
rec-4405-org	charles	green	38	salkauskas crescent	kela	dapto	4566	nsw	19480930	4365168
rec-1288-org	vanessa	parr	905	macquoid place	broadbridge manor	south grafton	2135	sa	19951119	9239102
rec-3585-org	mikayla	malloney	37	randwick road	avalind	hoppers crossing	4552	vic	19860208	7207688
...	...	...	...	...	...	...	...	...	...	...
rec-2153-org	annabel	grierson	97	mclachlan crescent	lantana lodge	broome	2480	nsw	19840224	7676186
rec-1604-org	sienna	musolino	22	smeaton circuit	pangani	mckinnon	2700	nsw	19890525	4971506
rec-1003-org	bradley	matthews	2	jondol place	horseshoe ck	jacobs well	7018	sa	19481122	8927667
rec-4883-org	brodee	egan	88	axon street	greenslopes	wamberal	2067	qld	19121113	6039042
rec-66-org	koula	houweling	3	mileham street	old airdmillan road	williamstown	2350	nsw	19440718	6375537

5000 rows × 10 columns

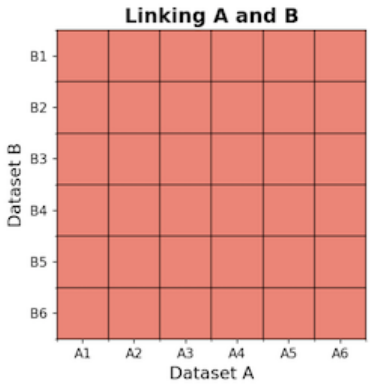
We notice that the records having the same numeric id represent the same entity.

```
given_name             michaela
surname                 neumann
street_number                 8
address_1        stanley street
address_2                miami
suburb            winston hills
postcode                  4223
state                      nsw
date_of_birth         19151111
soc_sec_id             5304218
Name: rec-1070-org, dtype: object
```

```
given_name             michafla
surname                 jakimow
street_number                 8
address_1        stanleykstreet
address_2                miami
suburb            winstonbhills
postcode                  4223
state                      NaN
date_of_birth         19151111
soc_sec_id             5304218
Name: rec-1070-dup-0, dtype: object
```

Indexing (Full)

Indexing is the process of creating all the possible links, also called **candidate links**, between the two datasets. In this specific example, we use a technique called **Full Indexing**, which returns the cartesian products of the records from each dataset.




```
WARNING:recordlinkage:indexing - performance warning - A full index can result in large number of r
ecord pairs.

WARNING:recordlinkage:indexing - performance warning - A full index can result in large number of r
ecord pairs.

MultiIndex([('rec-1070-org',  'rec-561-dup-0'),
            ('rec-1070-org', 'rec-2642-dup-0'),
            ('rec-1070-org',  'rec-608-dup-0'),
            ('rec-1070-org', 'rec-3239-dup-0'),
            ('rec-1070-org', 'rec-2886-dup-0'),
            ('rec-1070-org', 'rec-4285-dup-0'),
            ('rec-1070-org',  'rec-929-dup-0'),
            ('rec-1070-org', 'rec-4833-dup-0'),
            ('rec-1070-org',  'rec-717-dup-0'),
            ('rec-1070-org', 'rec-3984-dup-0'),
            ...
            (  'rec-66-org',  'rec-670-dup-0'),
            (  'rec-66-org', 'rec-4134-dup-0'),
            (  'rec-66-org', 'rec-3866-dup-0'),
            (  'rec-66-org', 'rec-3152-dup-0'),
            (  'rec-66-org', 'rec-3363-dup-0'),
            (  'rec-66-org', 'rec-4495-dup-0'),
            (  'rec-66-org', 'rec-4211-dup-0'),
            (  'rec-66-org', 'rec-3131-dup-0'),
            (  'rec-66-org', 'rec-3815-dup-0'),
            (  'rec-66-org',  'rec-493-dup-0')],
           names=['rec_id_1', 'rec_id_2'], length=25000000)
```

Comparison

Comparison refers to the process of evaluating all the possible candidate links in order to figure out all the attributes having the same value. In order to compare attributes, we need to specify (for each attribute):

- A **metric** to be used for evaluating the similarity between each pair of the same attributes of the different dataset
- A **threshold** to decide under which circumstances the metric shall return 1 or 0

The *similarity* is 1 in case of agreement or 0 in case of complete disagreement.

In this specific example we used the Jaro-Winkler method since, according to the documentation, it is faster than the Levenshtein distance and much faster than the Damerau-Levenshtein distance. Since comparing strings is computationally expensive, we chose a subset of attributes on which some errors are permitted.

```
comparison = {'surname':['jarowinkler',0.85],
              'address_1':['jarowinkler',0.85],
              'address_2':['jarowinkler',0.85] }
exact = ['given_name', 'date_of_birth', 'suburb', 'state']

features_full = compare_records(candidate_links_full, dfA, dfB, comparison, exact)
features_full
```

		surname	address_1	address_2	given_name	date_of_birth	suburb	state
rec_id_1	rec_id_2							
	rec-561-dup-0	0.0	0.0	0.0	0	0	0	0
	rec-2642-dup-0	0.0	0.0	0.0	0	0	0	1
rec-1070-org	rec-608-dup-0	0.0	0.0	0.0	0	0	0	0
	rec-3239-dup-0	0.0	0.0	0.0	0	0	0	0
	rec-2886-dup-0	0.0	0.0	0.0	0	0	0	0
...	...	...	...	...	...	...	...	...
	rec-4495-dup-0	0.0	0.0	0.0	0	0	0	1
	rec-4211-dup-0	0.0	0.0	0.0	0	0	0	0
rec-66-org	rec-3131-dup-0	0.0	0.0	0.0	0	0	0	0
	rec-3815-dup-0	0.0	0.0	0.0	0	0	0	0
	rec-493-dup-0	0.0	0.0	0.0	0	0	0	0

25000000 rows × 7 columns

Note: no cardinality reduction is employed. The output of indexing phase has the same cardinality of the comparison phase. The columns represent the results of the comparison.

Classification

In this phase, candidate links are classified into matches or non-matches. We analyze 3 different classification methods:

- **Deterministic approach** (no Machine Learning used)
- **Naive Bayes** (Supervised Learning)
- **Expectation-Conditional Maximisation** (Unsupervised Learning)

Deterministic approach

In the deterministic approach, we consider all the records that agree on at least a certain amount of attributes according to the rules we have previously specified.

```
7.0        1304
6.0        1860
5.0        1260
4.0         441
3.0         452
2.0       69727
1.0     5634539
0.0    19290417
dtype: int64
```

We notice that 1304 records have 7 common attributes, according to the comparison metrics that we chose. 1860 records have 6 common attributes instead, and so on.

We consider a match all the pairs that agree on at least 4 attributes. In the following table, the matches according to this rule are shown.

rec_id_1	rec_id_2	surname	address_1	address_2	given_name	date_of_birth	suburb	state
rec-1016-org	rec-1016-dup-0	1.0	1.0	0.0	1	1	1	0
rec-4405-org	rec-4405-dup-0	1.0	1.0	1.0	1	1	1	1
rec-1288-org	rec-1288-dup-0	1.0	1.0	1.0	1	1	0	1
rec-3585-org	rec-3585-dup-0	1.0	1.0	1.0	1	1	1	1
rec-298-org	rec-298-dup-0	1.0	0.0	0.0	1	1	1	1
...	...	...	...	...	...	...	...	...
rec-2153-org	rec-2153-dup-0	1.0	0.0	0.0	0	1	1	1
rec-1604-org	rec-1604-dup-0	0.0	1.0	1.0	0	1	1	1
rec-1003-org	rec-1003-dup-0	0.0	1.0	1.0	0	1	0	1
rec-4883-org	rec-4883-dup-0	1.0	1.0	1.0	1	1	1	1
rec-66-org	rec-66-dup-0	1.0	1.0	1.0	1	1	1	1

4865 rows × 7 columns

Naive Bayes

Naive Bayes is a supervised-learning algorithm that provides a method for computing the probability of a given hypotesis on prior knowledge. It is based on the assumption that all features are conditionally independent.

We initialise the NaiveBayesClassifier.

```
nb_full = rl.NaiveBayesClassifier()
nb_full.fit(features_full, true_links)
```

We print also the parameters that are trained.

- p is the probability P(Match)
- m is the probability that P(x_i=1|Match)
- u is the probability that P(x_i=1|Non-Match)

```
p probability P(Match): 0.00019999999999999985

m probabilities P(x_i=1|Match): {'surname': {0.0: 0.14700001411999955, 1.0: 0.8529999858800007}, 'a
ddress_1': {0.0: 0.11280001548799944, 1.0: 0.8871999845120011}, 'address_2': {0.0: 0.23020001079199
967, 1.0: 0.7697999892080011}, 'given_name': {0.0: 0.3426000062959999, 1.0: 0.6573999937040006}, 'd
ate_of_birth': {0.0: 0.10620001575199943, 1.0: 0.8937999842480008}, 'suburb': {0.0: 0.2542000098319
9964, 1.0: 0.7457999901680004}, 'state': {0.0: 0.05860001765599933, 1.0: 0.9413999823440014}}

u probabilities P(x_i=1|Non-Match): {'surname': {0.0: 0.9951189437847974, 1.0: 0.00488105621520399
8}, 'address_1': {0.0: 0.9971519903941033, 1.0: 0.0028480096058983977}, 'address_2': {0.0: 0.998770
9141788471, 1.0: 0.0012290858211543999}, 'given_name': {0.0: 0.9970409281816627, 1.0: 0.00295907181
8339998}, 'date_of_birth': {0.0: 0.9999744748909813, 1.0: 2.5525109021600138e-05}, 'suburb': {0.0:
0.9991115823124727, 1.0: 0.0008884176875304002}, 'state': {0.0: 0.7817865973172118, 1.0: 0.21821340
268279102}}

weights of features: {'surname': {0.0: 0.14772104886367185, 1.0: 174.75725504307687}, 'address_1':
{0.0: 0.11312218856768025, 1.0: 311.5157977959615}, 'address_2': {0.0: 0.230483294541333, 1.0: 626.
3191519734385}, 'given_name': {0.0: 0.34361679306466497, 1.0: 222.1642575991257}, 'date_of_birth':
{0.0: 0.10620272658817369, 1.0: 35016.50016427509}, 'suburb': {0.0: 0.2544260464318173, 1.0: 839.46
99932653922}, 'state': {0.0: 0.07495653910810424, 1.0: 4.31412539637852}}
```

We tried to predict the matches starting by the output of the comparison phase, using the trained model. We know from the documentation that the total matches are 5000 and our model found 4958.

```
Predicted number of links: 4958
```

We predict the match probability for each pair in the dataset, since we apply the Full Index method.

```
probs_bayes_full = nb_full.prob(features_full)
probs_bayes_full
```

```
rec_id_1      rec_id_2
rec-1070-org  rec-561-dup-0     5.362010e-10
              rec-2642-dup-0    3.086106e-08
              rec-608-dup-0     5.362010e-10
              rec-3239-dup-0    5.362010e-10
              rec-2886-dup-0    5.362010e-10
                                    ...
rec-66-org    rec-4495-dup-0    3.086106e-08
              rec-4211-dup-0    5.362010e-10
              rec-3131-dup-0    5.362010e-10
              rec-3815-dup-0    5.362010e-10
              rec-493-dup-0     5.362010e-10
Length: 25000000, dtype: float64
```

Expectation-Conditional Maximisation

Unsupervised learning with the ECM algorithm. This classifier doesn't need training label since it is often hard to collect.

We initialise the Expectation-Conditional Maximisation classifier.

```
ecm_full = rl.ECMClassifier()
ecm_full.fit(features_full)
```

We print the parameters that are trained (m, u and p).

```
p probability P(Match): 0.00019964038993562045

m probabilities P(x_i=1|Match): {'surname': {0.0: 0.14559845395438062, 1.0: 0.8544015460456198}, 'a
ddress_1': {0.0: 0.11167423011028436, 1.0: 0.8883257698897149}, 'address_2': {0.0: 0.22930633589775
026, 1.0: 0.7706936641022502}, 'given_name': {0.0: 0.3415113651654586, 1.0: 0.6584886348345418}, 'd
ate_of_birth': {0.0: 0.10497269467212568, 1.0: 0.8950273053278734}, 'suburb': {0.0: 0.2537178842231
8436, 1.0: 0.7462821157768156}, 'state': {0.0: 0.05893559457665887, 1.0: 0.9410644054233418}}

u probabilities P(x_i=1|Non-Match): {'surname': {0.0: 0.9951189185968193, 1.0: 0.004881081403180484
4}, 'address_1': {0.0: 0.9971518971067167, 1.0: 0.002848102893281545}, 'address_2': {0.0: 0.9987708
161888874, 1.0: 0.0012291838111130658}, 'given_name': {0.0: 0.9970409101739776, 1.0: 0.002959089826
0201783}, 'date_of_birth': {0.0: 0.9999743984891409, 1.0: 2.5601510856925678e-05}, 'suburb': {0.0:
0.9991114106542701, 1.0: 0.0008885893457278089}, 'state': {0.0: 0.7817862701907439, 1.0: 0.21821372
980925488}}

weights of features: {'surname': {0.0: 0.1463126177519403, 1.0: 175.043494560221l8}, 'address_1':
{0.0: 0.11199319826228323, 1.0: 311.9008698685735}, 'address_2': {0.0: 0.22958854241730656, 1.0: 62
6.996269503714}, 'given_name': {0.0: 0.3425249271926735, 1.0: 222.53080289900313}, 'date_of_birth':
{0.0: 0.10497538220051303, 1.0: 34959.94085387161}, 'suburb': {0.0: 0.253943535743463, 1.0: 839.850
3981223914}, 'state': {0.0: 0.07538581428691435, 1.0: 4.312581093068459}}


Predicted number of links: 4958


rec_id_1      rec_id_2
rec-1070-org  rec-561-dup-0     5.170889e-10
              rec-2642-dup-0    2.958100e-08
              rec-608-dup-0     5.170889e-10
              rec-3239-dup-0    5.170889e-10
              rec-2886-dup-0    5.170889e-10
                                    ...
rec-66-org    rec-4495-dup-0    2.958100e-08
              rec-4211-dup-0    5.170889e-10
              rec-3131-dup-0    5.170889e-10
              rec-3815-dup-0    5.170889e-10
              rec-493-dup-0     5.170889e-10
Length: 25000000, dtype: float64
```

Evaluation

We compare the results of the deterministic method and the two ML-based ones (Naive Bayes and ECM)

$$
\begin{aligned}
precision &= \frac{TP}{TP + FP} \\
recall &= \frac{TP}{TP + FN} \\
F1 &= \frac{2 \times precision \times recall}{precision + recall}
\end{aligned}
$$

```
Deterministic approach

Confusion matrix:
 [[    4865      135]
 [       0 24995000]]
Fscore: 0.9863152559553979
Recall: 0.973
Precision: 1.0


Naive Bayes

Confusion matrix:
 [[    4954       46]
 [       4 24994996]]
Fscore: 0.9949789114279977
Recall: 0.9908
Precision: 0.9991932230738201


Expectation-Conditional Maximisation

Confusion matrix:
 [[    4954       46]
 [       4 24994996]]
Fscore: 0.9949789114279977
Recall: 0.9908
Precision: 0.9991932230738201
```

Part 2 : Using a Block Index

Indexing (Block)

Indexing is the process of creating all the possible links between the two datasets. In this specific example, we use a technique called **Blocking**, which groups together all the records that agree on AT LEAST one of the specified attributes.

We have decided to use **surname**, **date_of_birth** and **soc_sec_id** as discriminating variables. By doing that, the resulting records will agree on at least one of these attributes.

Note: the cardinality using the Block Index method is way less than the Full Index method.

```
MultiIndex([(  'rec-0-org',    'rec-0-dup-0'),
            (  'rec-0-org', 'rec-1505-dup-0'),
            (  'rec-0-org', 'rec-1636-dup-0'),
            (  'rec-0-org', 'rec-2074-dup-0'),
            (  'rec-0-org', 'rec-2683-dup-0'),
            (  'rec-0-org', 'rec-2724-dup-0'),
            (  'rec-0-org', 'rec-2894-dup-0'),
            (  'rec-1-org',    'rec-1-dup-0'),
            (  'rec-1-org', 'rec-1052-dup-0'),
            (  'rec-1-org', 'rec-2552-dup-0'),
            ...
            ('rec-999-org', 'rec-3685-dup-0'),
            ('rec-999-org',  'rec-370-dup-0'),
            ('rec-999-org', 'rec-3766-dup-0'),
            ('rec-999-org', 'rec-3862-dup-0'),
            ('rec-999-org', 'rec-3913-dup-0'),
            ('rec-999-org', 'rec-3940-dup-0'),
            ('rec-999-org', 'rec-4941-dup-0'),
            ('rec-999-org',  'rec-859-dup-0'),
            ('rec-999-org',  'rec-911-dup-0'),
            ('rec-999-org',  'rec-999-dup-0')],
           names=['rec_id_1', 'rec_id_2'], length=87132)
```

We verify that a pair of candidate links agree on the given attributes:

```
given_name               rachael
surname                     dent
street_number                  1
address_1            knox street
address_2        lakewood estate
suburb                     byford
postcode                    4129
state                        vic
date_of_birth           19280722
soc_sec_id               1683994
Name: rec-0-org, dtype: object
```

```
given_name               rachael
surname                     dent
street_number                  4
address_1            knox street
address_2        lakewood estate
suburb                     byford
postcode                    4129
state                        vic
date_of_birth           19280722
soc_sec_id               1683994
Name: rec-0-dup-0, dtype: object
```

Comparison

We used the same metrics and threshold for each pair of attributes used in the previous experiment.

		surname	address_1	address_2	given_name	date_of_birth	suburb	state
rec_id_1	rec_id_2							
	rec-0-dup-0	1.0	1.0	1.0	1	1	1	1
	rec-1505-dup-0	1.0	0.0	0.0	0	0	0	0
rec-0-org	rec-1636-dup-0	1.0	0.0	0.0	0	0	0	0
	rec-2074-dup-0	1.0	0.0	0.0	0	0	0	1
	rec-2683-dup-0	1.0	0.0	0.0	0	0	0	0
...	...	...	...	...	...	...	...	...
	rec-3940-dup-0	1.0	0.0	0.0	0	0	0	0
	rec-4941-dup-0	1.0	0.0	0.0	0	0	0	0
rec-999-org	rec-859-dup-0	1.0	0.0	0.0	0	0	0	0
	rec-911-dup-0	1.0	0.0	0.0	0	0	0	1
	rec-999-dup-0	1.0	1.0	1.0	1	1	1	1

87132 rows × 7 columns

Classification

Deterministic approach

Also in this case, we consider a match all the pairs that agree on at least 4 attributes.

```
7.0     1304
6.0     1857
5.0     1257
4.0      438
3.0      268
2.0    18304
1.0    63704
dtype: int64
```

Naive Bayes

```
# Initialise the NaiveBayesClassifier.
nb_block = rl.NaiveBayesClassifier()
nb_block.fit(features_block, true_links)
```

```
p probability P(Match): 0.05726943028967549

m probabilities P(x_i=1|Match): {'surname': {0.0: 0.1462925993469899, 1.0: 0.8537074006530102}, 'ad
dress_1': {0.0: 0.11302606761418561, 1.0: 0.8869739323858145}, 'address_2': {0.0: 0.230260531853285
24, 1.0: 0.7697394681467138}, 'given_name': {0.0: 0.34268537704667806, 1.0: 0.6573146229533218}, 'd
ate_of_birth': {0.0: 0.10440883349062785, 1.0: 0.8955911665093714}, 'suburb': {0.0: 0.2543086270818
1856, 1.0: 0.7456913729181811}, 'state': {0.0: 0.05871745255641471, 1.0: 0.941282547443585}}

u probabilities P(x_i=1|Non-Match): {'surname': {0.0: 0.0077305166474385435, 1.0: 0.992269483352561
5}, 'address_1': {0.0: 0.9970904032112915, 1.0: 0.002909596788708342}, 'address_2': {0.0: 0.9987338
986176765, 1.0: 0.001266101382323047}, 'given_name': {0.0: 0.9970173589710086, 1.0: 0.0029826410289
92133}, 'date_of_birth': {0.0: 0.9922329612324204, 1.0: 0.007767038767580439}, 'suburb': {0.0: 0.99
90382496188591, 1.0: 0.0009617503811405855}, 'state': {0.0: 0.7818777232551495, 1.0: 0.218122276744
85082}}

weights of features: {'surname': {0.0: 18.924039106165434, 1.0: 0.8603584157084078}, 'address_1':
{0.0: 0.11335588754055481, 1.0: 304.8442780209312}, 'address_2': {0.0: 0.2305524346094423, 1.0: 60
7.960372600173}, 'given_name': {0.0: 0.3437105422119764, 1.0: 220.38006470240086}, 'date_of_birth':
{0.0: 0.10522612891325946, 1.0: 115.30664302173464}, 'suburb': {0.0: 0.2545534439535616, 1.0: 775.3
481439059377}, 'state': {0.0: 0.07509799909883542, 1.0: 4.315389337993445}}


Predicted number of links: 5107


rec_id_1     rec_id_2
rec-0-org    rec-0-dup-0       1.000000e+00
             rec-1505-dup-0    9.443910e-07
             rec-1636-dup-0    9.443910e-07
             rec-2074-dup-0    5.426506e-05
             rec-2683-dup-0    9.443910e-07
                                    ...
rec-999-org  rec-3940-dup-0    9.443910e-07
             rec-4941-dup-0    9.443910e-07
             rec-859-dup-0     9.443910e-07
             rec-911-dup-0     5.426506e-05
             rec-999-dup-0     1.000000e+00
Length: 87132, dtype: float64
```

Expectation-Conditional Maximisation

```
# Initialise the Expectation-Conditional Maximisation classifier.
ecm_block = rl.ECMClassifier()
ecm_block.fit(features_block)
```

```
p probability P(Match): 0.05890218346289975

m probabilities P(x_i=1|Match): {'surname': {0.0: 0.17208594081201783, 1.0: 0.8279140591879811}, 'a
ddress_1': {0.0: 0.1364062789961936, 1.0: 0.863593721003806}, 'address_2': {0.0: 0.2507244037219689
7, 1.0: 0.7492755962780303}, 'given_name': {0.0: 0.36109752267741557, 1.0: 0.6389024773225834}, 'da
te_of_birth': {0.0: 0.1019492644528454, 1.0: 0.8980507355471536}, 'suburb': {0.0: 0.275488967249009
1, 1.0: 0.7245110327509904}, 'state': {0.0: 0.06435714759255907, 1.0: 0.9356428524074404}}

u probabilities P(x_i=1|Non-Match): {'surname': {0.0: 0.00587574265639725, 1.0: 0.994124257343602
7}, 'address_1': {0.0: 0.9971608673018983, 1.0: 0.002839132698101932}, 'address_2': {0.0: 0.9987863
491894582, 1.0: 0.001213650810542423}, 'given_name': {0.0: 0.9970001958451015, 1.0: 0.0029998041548
98576}, 'date_of_birth': {0.0: 0.9939272291845378, 1.0: 0.006072770815462413}, 'suburb': {0.0: 0.99
9004663262730 1, 1.0: 0.0009953367372699084}, 'state': {0.0: 0.7827793845853916, 1.0: 0.217220615414
60944}}

weights of features: {'surname': {0.0: 29.287521744108087, 1.0: 0.832807421277747}, 'address_1':
{0.0: 0.13679465718031986, 1.0: 304.1751875779358}, 'address_2': {0.0: 0.25102906535060127, 1.0: 61
7.3732920288273}, 'given_name': {0.0: 0.3621840037567228, 1.0: 212.98139622857641}, 'date_of_birt
h': {0.0: 0.10257216168279153, 1.0: 147.88154581110624}, 'suburb': {0.0: 0.2757634447363513, 1.0: 7
27.9054470934519}, 'state': {0.0: 0.08221620147373528, 1.0: 4.307339110616076}}


Predicted number of links: 5110
```

```
rec_id_1     rec_id_2
rec-0-org    rec-0-dup-0       1.000000
             rec-1505-dup-0    0.000002
             rec-1636-dup-0    0.000002
             rec-2074-dup-0    0.000079
             rec-2683-dup-0    0.000002
                                 ...
rec-999-org  rec-3940-dup-0    0.000002
             rec-4941-dup-0    0.000002
             rec-859-dup-0     0.000002
             rec-911-dup-0     0.000079
             rec-999-dup-0     1.000000
Length: 87132, dtype: float64
```

Evaluation

```
Deterministic approach

Confusion matrix:
 [[ 4856   144]
 [    0 82132]]
Fscore: 0.9853896103896103
Recall: 0.9712
Precision: 1.0


Naive Bayes

Confusion matrix:
 [[ 4956    44]
 [  151 81981]]
Fscore: 0.9807064410804394
Recall: 0.9912
Precision: 0.9704327393773252


Expectation-Conditional Maximisation

Confusion matrix:
 [[ 4959    41]
 [  151 81981]]
Fscore: 0.9810089020771513
Recall: 0.9918
Precision: 0.9704500978473581
```

Results Analysis

From the previous two experiments, we noticed that:

- The cardinality of the candidate links with Full Index method is 25000000, with the Block Index is 87132.
- This leads to a significative difference in the execution time:
  - The Full Index takes 160 seconds circa for the comparison phase; the Block Index takes about 0.56 seconds instead.
  - Naive Bayes takes about 150s with Full Index and 0.23s with Block Index.
  - ECM takes about 92s with Full Index and 0.29s with Block Index.
- This does not lead to a significative difference in the evaluation results:
  - The Full Index method brings a very small increase in the Fscore.

We do not consider this result to be meaningful. Blocking should always be preferred, provided that it is clear what attributes should be used in the index.

Experiment 2: Training and Testing our models for evaluating re-usability

In this experiment, we first train our models on a dataset. Then, we test the models on another one with the same attributes but with different records and a different duplicate distribution. We do that in order to investigate the *re-usability* of the models we create, and to evaluate their applications in real-case scenarios.

Part 1 : Training

Dataset description

This data set contains 5000 records (4000 originals and 1000 duplicates), with a maximum of 5 duplicates based on one original record (and a poisson distribution of duplicate records). Distribution of duplicates: 19 originals records have 5 duplicate records 47 originals records have 4 duplicate records 107 originals records have 3 duplicate records 141 originals records have 2 duplicate records 114 originals records have 1 duplicate record 572 originals records have no duplicate record.

```
Loading data...
5000 records in dataset A
1934 links between dataset A and A
```

rec_id	given_name	surname	street_number	address_1	address_2	suburb	postcode	state	date_of_birth	soc_sec_id
rec-2778-org	sarah	bruhn	44	forbes street	wintersloe	kellerberrin	4510	vic	19300213	7535316
rec-712-dup-0	jacob	lanyon	5	milne cove	wellwod	beaconsfield upper	2602	vic	19080712	9497788
rec-1321-org	brinley	efthimiou	35	sturdee crescent	tremearne	scarborough	5211	qld	19940319	6814956
rec-3004-org	aleisha	hobson	54	oliver street	inglewood	toowoomba	3175	qld	19290427	5967384
rec-1384-org	ethan	gazzola	49	sheaffe street	bimby vale	port pirie	3088	sa	19631225	3832742
...	...	...	...	...	...	...	...	...	...	...
rec-1487-org	thomas	green	44	tuthill place	holmeleigh	bonny hills	4740	vic	19420210	9334580
rec-1856-org	james	mcneill	42	archibald street	NaN	evans head	2250	nsw	19011207	4837378
rec-3307-org	paige	lock	7	a'beckett street	camboon	carina heights	2290	nsw	19871002	5142242
rec-227-org	antonio	collier	25	govett place	the rocks	broken hill	2304	qld	19400225	3973395
rec-1143-org	harry	ryan	30	van raalte place	anana	preston west	2572	wa	19250425	6392122

5000 rows × 10 columns

We notice that the records having the same numeric id represent the same entity but with some errors in various attributes.

```
given_name                   jacob
surname                     lanyon
street_number                    5
address_1                milne cove
address_2                  wellwood
suburb           beaconsfield upper
postcode                      2602
state                          vic
date_of_birth             19080712
soc_sec_id                 9497788
Name: rec-712-org, dtype: object
```

```
given_name                   jacob
surname                     lanyon
street_number                    5
address_1                milne cove
address_2                  wellwod
suburb           beaconsfield upper
postcode                      2602
state                          vic
date_of_birth             19080712
soc_sec_id                 9497788
Name: rec-712-dup-0, dtype: object
```

```
given_name                   jacob
surname                     lanyln
street_number                    5
address_1               milne cove
address_2                 wellwood
suburb           beaconsfieod upper
postcode                      2602
state                          vic
date_of_birth             19080712
soc_sec_id                 9497788
Name: rec-712-dup-1, dtype: object
```

Indexing (Block)

We have decided to use **surname**, **date_of_birth** and **soc_sec_id** as discriminating variables. It's important to notice that the discriminating variables are the same throughout different experiments.

```
MultiIndex([(    'rec-0-org', 'rec-3741-dup-0'),
            (    'rec-0-org',   'rec-3741-org'),
            (   'rec-10-org',   'rec-2162-org'),
            (   'rec-10-org',    'rec-343-org'),
            (   'rec-10-org',    'rec-956-org'),
            ('rec-100-dup-0',    'rec-100-org'),
            ('rec-100-dup-1',  'rec-100-dup-0'),
            ('rec-100-dup-1',    'rec-100-org'),
            ('rec-100-dup-2',  'rec-100-dup-0'),
            ('rec-100-dup-2',  'rec-100-dup-1'),
            ...
            (  'rec-998-org',   'rec-1578-org'),
            (  'rec-998-org',   'rec-2112-org'),
            (  'rec-998-org',   'rec-2273-org'),
            (  'rec-998-org',   'rec-2670-org'),
            (  'rec-998-org',    'rec-305-org'),
            (  'rec-998-org',    'rec-345-org'),
            (  'rec-998-org',   'rec-3803-org'),
            (  'rec-998-org',   'rec-3936-org'),
            (  'rec-998-org',    'rec-622-org'),
            (  'rec-999-org',   'rec-3297-org')],
           names=['rec_id_1', 'rec_id_2'], length=50929)
```

We show a pair of candidate links in order to verify that at least one of the values of the discriminating variables are equal.

```
In [100]: df.loc[candidate_links[0][0]]
```

```
Out[100]: given_name              annabelle
          surname                  friswell
          street_number                 205
          address_1           meares place
          address_2        sunningdale farm
          suburb              wildes meadow
          postcode                     7018
          state                          wa
          date_of_birth            19761129
          soc_sec_id                9016980
          Name: rec-0-org, dtype: object
```

```
In [101]: df.loc[candidate_links[0][1]]
```

```
Out[101]: given_name             michael
          surname                shepherd
          street_number                70
          address_1         hurry nplace
          address_2             tintagel
          suburb                 penguin
          postcode                  4179
          state                      vic
          date_of_birth          19761129
          soc_sec_id              8270855
          Name: rec-3741-dup-0, dtype: object
```

Comparison

We used the same metrics and threshold used in the first experiment.

		surname	address_1	address_2	given_name	date_of_birth	suburb	state
rec_id_1	rec_id_2							
rec-0-org	rec-3741-dup-0	0.0	0.0	0.0	0	1	0	0
	rec-3741-org	0.0	0.0	0.0	0	1	0	0
	rec-2162-org	1.0	0.0	0.0	0	0	0	0
rec-10-org	rec-343-org	1.0	0.0	0.0	0	0	0	0
	rec-956-org	1.0	0.0	0.0	0	0	0	0
...	...	...	...	...	...	...	...	...
rec-998-org	rec-345-org	1.0	0.0	0.0	0	0	0	1
	rec-3803-org	1.0	0.0	0.0	0	0	0	1
	rec-3936-org	1.0	0.0	0.0	0	0	0	0
	rec-622-org	1.0	0.0	0.0	0	0	0	0
rec-999-org	rec-3297-org	1.0	0.0	0.0	0	0	0	0

50929 rows × 7 columns

Classification

We train our ML models in order to acquire knowledge to classify records in the test set, considering the weights of the attributes.

Naive Bayes

```
# Initialise the NaiveBayesClassifier.
nb = rl.NaiveBayesClassifier()
nb.fit(features, true_links)
```

```
p probability P(Match): 0.03769954250034364

m probabilities P(x_i=1|Match): {'surname': {0.0: 0.19739586485459734, 1.0: 0.8026041351454025}, 'a
ddress_1': {0.0: 0.12031253955077714, 1.0: 0.879687460449223}, 'address_2': {0.0: 0.298437520996091
6, 1.0: 0.7015624790039081}, 'given_name': {0.0: 0.3880208449978287, 1.0: 0.6119791550021711}, 'dat
e_of_birth': {0.0: 0.13072920513237446, 1.0: 0.8692707948676252}, 'suburb': {0.0: 0.3546875151367171
5, 1.0: 0.645312484863283}, 'state': {0.0: 0.06197921229383207, 1.0: 0.9380207877061676}}

u probabilities P(x_i=1|Non-Match): {'surname': {0.0: 0.006304966409006646, 1.0: 0.993695033590994
5}, 'address_1': {0.0: 0.9959803281194052, 1.0: 0.004019671880594699}, 'address_2': {0.0: 0.9986533
065410298, 1.0: 0.0013466934589700125}, 'given_name': {0.0: 0.9961435634428644, 1.0: 0.003856436557
136706}, 'date_of_birth': {0.0: 0.9936746291755609, 1.0: 0.006325370824438895}, 'suburb': {0.0: 0.9
9904099043424 37, 1.0: 0.0009590095657572717}, 'state': {0.0: 0.7887734894049123, 1.0: 0.21122651059
50886}}

weights of features: {'surname': {0.0: 31.3079962761764, 1.0: 0.8076966352996334}, 'address_1': {0.
0: 0.12079810830997982, 1.0: 218.84558903824646}, 'address_2': {0.0: 0.29883996682469327, 1.0: 520.
9518724034511}, 'given_name': {0.0: 0.38952301579578924, 1.0: 158.6903209569583}, 'date_of_birth':
{0.0: 0.13156137964480266, 1.0: 137.42606069972751}, 'suburb': {0.0: 0.3550279903755986, 1.0: 672.8
947321330616}, 'state': {0.0: 0.07857669296237643, 1.0: 4.440828876372956}}
```

Expectation-Conditional Maximisation

```
# Initialise the Expectation-Conditional Maximisation classifier.
ecm = rl.ECMClassifier()
ecm.fit(features)
```

```
p probability P(Match): 0.04422886741322979

m probabilities P(x_i=1|Match): {'surname': {0.0: 0.30542655786279255, 1.0: 0.694573442137208}, 'ad
dress_1': {0.0: 0.2468825894881691, 1.0: 0.7531174105118315}, 'address_2': {0.0: 0.4004723999183774
6, 1.0: 0.5995276000816229}, 'given_name': {0.0: 0.4751710465591838, 1.0: 0.5248289534408166}, 'dat
e_of_birth': {0.0: 0.12148308903836183, 1.0: 0.8785169109616388}, 'suburb': {0.0: 0.448398605539636
23, 1.0: 0.551601394460364}, 'state': {0.0: 0.1612905703295032, 1.0: 0.8387094296704972}}

u probabilities P(x_i=1|Non-Match): {'surname': {0.0: 3.4736045678538596e-07, 1.0: 0.99999965263954
13}, 'address_1': {0.0: 0.9961053266705839, 1.0: 0.003894673329415 1813}, 'address_2': {0.0: 0.99871
50895414244, 1.0: 0.0012849104585748773}, 'given_name': {0.0: 0.9962650126166345, 1.0: 0.0037349873
83363294}, 'date_of_birth': {0.0: 0.9999976876787589, 1.0: 2.3123212408447907e-06}, 'suburb': {0.0:
0.9991063399654829, 1.0: 0.0008936600345149506}, 'state': {0.0: 0.7891428733450196, 1.0: 0.21085712
665497933}}

weights of features: {'surname': {0.0: 879278.4322353012, 1.0: 0.6945736834046412}, 'address_1':
{0.0: 0.24784787600058097, 1.0: 193.37113714359157}, 'address_2': {0.0: 0.4009876331219353, 1.0: 46
6.59095665434324}, 'given_name': {0.0: 0.47695245797217506, 1.0: 140.51692805671993}, 'date_of_birt
h': {0.0: 0.1214833699469386, 1.0: 379928.57369622134}, 'suburb': {0.0: 0.4487996798770464, 1.0: 61
7.2385170606351}, 'state': {0.0: 0.20438703278891002, 1.0: 3.97761955204321}}
```

Part 2 : Testing

Dataset description

This data set contains 5000 records (2000 originals and 3000 duplicates), with a maximum of 5 duplicates based on one original record (and a Zipf distribution of duplicate records). Distribution of duplicates: 168 originals records have 5 duplicate records 161 originals records have 4 duplicate records 212 originals records have 3 duplicate records 256 originals records have 2 duplicate records 368 originals records have 1 duplicate record 1835 originals records have no duplicate record.

```
Loading data...
5000 records in dataset A
6538 links between dataset A and A
```

rec_id	given_name	surname	street_number	address_1	address_2	suburb	postcode	state	date_of_birth	soc_sec_id
rec-1496-org	mitchell	green	7	wallaby place	delmar	cleveland	2119	sa	19560409	1804974
rec-552-dup-3	harley	mccarthy	177	pridhamstreet	milton	marsden	3165	nsw	19080419	6089216
rec-988-dup-1	madeline	mason	54	hoseason street	lakefront retrmnt vlge	granville	4881	nsw	19081128	2185997
rec-1716-dup-1	isabelle	NaN	23	gundulu place	currin ga	utakarra	2193	wa	19921119	4314184
rec-1213-org	taylor	hathaway	7	yuranigh court	brentwood vlge	NaN	4220	nsw	19991207	9144092
...	...	...	...	...	...	...	...	...	...	...
rec-937-org	jack	campbell	169	marr street	rhosewyn	oakleigh	3356	vic	19770109	1485686
rec-1200-dup-0	william	lazaroff	12	leah ylose	milwlood	forbes	7256	qld	NaN	8072193
rec-1756-org	destynii	bowerman	12	halford crescent	sutton	nollamara	2431	qld	19880821	6089424
rec-1444-org	gianni	dooley	38	ashburton circuit	brentwood vlge	ryde	6025	qld	19371212	5854405
rec-993-dup-0	jake	westbrook	231	booroondar a street	jodayne	salisbury east	2074	nsw	19001115	2330929

5000 rows × 10 columns

Indexing

```
candidate_links = indexing_dataset(df, None, ['surname','date_of_birth', 'soc_sec_id'])
candidate_links
```

```
MultiIndex([(   'rec-1-org', 'rec-1963-dup-0'),
            (   'rec-1-org',  'rec-567-dup-1'),
            (   'rec-1-org',  'rec-910-dup-1'),
            ('rec-10-dup-0',   'rec-10-dup-2'),
            ('rec-10-dup-1',   'rec-10-dup-0'),
            ('rec-10-dup-1',   'rec-10-dup-2'),
            ('rec-10-dup-1', 'rec-1411-dup-0'),
            ('rec-10-dup-1', 'rec-1411-dup-1'),
            ('rec-10-dup-1', 'rec-1411-dup-2'),
            ('rec-10-dup-1',   'rec-1411-org'),
            ...
            ( 'rec-997-org',   'rec-66-dup-0'),
            ( 'rec-997-org',     'rec-66-org'),
            ( 'rec-997-org',  'rec-724-dup-0'),
            ( 'rec-997-org',  'rec-724-dup-4'),
            ( 'rec-997-org',    'rec-724-org'),
            ( 'rec-997-org',    'rec-849-org'),
            ( 'rec-997-org',  'rec-997-dup-0'),
            ( 'rec-999-org',    'rec-234-org'),
            ( 'rec-999-org',  'rec-679-dup-1'),
            ( 'rec-999-org',    'rec-679-org')],
           names=['rec_id_1', 'rec_id_2'], length=40470)
```

Comparison

Also in this case, we used the same metrics and threshold used in the previous experiments.

		surname	address_1	address_2	given_name	date_of_birth	suburb	state
rec_id_1	**rec_id_2**							
	rec-1963-dup-0	1.0	0.0	0.0	0	0	0	0
rec-1-org	rec-567-dup-1	1.0	0.0	0.0	0	0	0	1
	rec-910-dup-1	1.0	0.0	0.0	0	0	0	1
rec-10-dup-0	rec-10-dup-2	0.0	0.0	1.0	0	1	1	1
rec-10-dup-1	rec-10-dup-0	0.0	0.0	1.0	0	1	1	1
...	...	...	...	...	...	...	...	...
rec-997-org	rec-849-org	1.0	0.0	0.0	0	0	0	1
	rec-997-dup-0	1.0	1.0	1.0	0	1	1	1
	rec-234-org	1.0	0.0	0.0	0	0	0	0
rec-999-org	rec-679-dup-1	1.0	0.0	0.0	0	0	0	0
	rec-679-org	1.0	0.0	0.0	0	0	0	0

40470 rows × 7 columns

Classification

Deterministic approach

We consider a match all the pairs that agree on at least 4 attributes.

```
7.0      969
6.0     2066
5.0     1955
4.0     1007
3.0      451
2.0     6929
1.0    27093
dtype: int64
```

		surname	address_1	address_2	given_name	date_of_birth	suburb	state
rec_id_1	rec_id_2							
rec-10-dup-0	rec-10-dup-2	0.0	0.0	1.0	0	1	1	1
rec-10-dup-1	rec-10-dup-0	0.0	0.0	1.0	0	1	1	1
	rec-10-dup-2	1.0	0.0	1.0	0	1	1	1
rec-10-org	rec-10-dup-0	0.0	1.0	1.0	0	1	1	1
	rec-10-dup-1	1.0	0.0	1.0	0	1	1	1
...	...	...	...	...	...	...	...	...
rec-995-dup-0	rec-995-org	1.0	1.0	1.0	1	1	1	1
rec-996-dup-1	rec-996-dup-0	1.0	1.0	1.0	0	1	0	1
rec-996-org	rec-996-dup-0	1.0	1.0	1.0	0	1	1	1
	rec-996-dup-1	1.0	1.0	1.0	1	1	0	1
rec-997-org	rec-997-dup-0	1.0	1.0	1.0	0	1	1	1

5997 rows × 7 columns

Naive Bayes

```
Predicted number of links: 6428
```

Expectation-Conditional Maximisation

```
Predicted number of links: 6750
```

Evaluation

```
Deterministic approach

Confusion matrix:
 [[ 5997   541]
 [    0 33932]]
Fscore: 0.9568408456322298
Recall: 0.9172529825634751
Precision: 1.0


Naive Bayes

Confusion matrix:
 [[ 6364   174]
 [   64 33868]]
Fscore: 0.9816443004781737
Recall: 0.9733863566840012
Precision: 0.9900435594275047


Expectation-Conditional Maximisation

Confusion matrix:
 [[ 6436   102]
 [  314 33618]]
Fscore: 0.9686935580975315
Recall: 0.9843988987457938
Precision: 0.9534814814814815
```

Results Analysis

While the results of the Confusion Matrices may be a little flawed due to the "toy-like" nature of our datasets, we think that the approach can work in real-case scenarios. However, it does require:

- a large amount of data (which is not a problem anymore)
- high computational power (again, not a problem anymore)
- the **true links**, which may not be always available in pratice. While it may be easy for humans to create true links, the sheer amount of time required for datasets with millions (or more) of records may be too high. The key to solving these problem may be to investigate the usage of unsupervised methods for non labelled problems, so that true links are not required in the first place.